7. VV&E for Small Expert Systems

From "Verification, Validation, and Evaluation of Expert Systems, Volume I"

Small expert systems are those for which direct proof of completeness, consistency, and specification satisfaction are practical without partitioning the knowledge base. This chapter discusses techniques for these proofs.

The basic method for verifying properties of small systems is:

1. Represent the property to be verified as a logical formula.

2. Verify the logical formula using one of the following techniques:

- Verify the formula on a case-by-case basis, e.g., by checking each Hoffman Region.
- Apply Boolean algebra simplifications to verify the formula.

Completeness

To verify completeness the user must demonstrate that for all inputs, the expert system produces *some* conclusion. This is done by:

1. Constructing a logical formula that represents conditions under which the system is complete; this logical formula will be called the *completeness formula*.

2. Showing that the truth value of that formula is TRUE.

If the expert system E is C*artesian*, i.e,. it is required to produce values of more than one variable, then the completeness formula for E is the AND of the completeness formulas for systems which set *each* of the required output variables for E.

For a system E that is required to take one of some set of actions a pseudo-variable is created of which values are the enumerated set of acceptable actions. Then the completeness formula for E is the completeness formula for a system that outputs the value of this variable.

The completeness formula for an expert system in E which sets a single variable v is constructed by an iterative substitution process from an initial formula. That initial formula is:

$$(v = e1) \text{ OR } (v = e2) \text{ OR } ... \text{ OR } (v = en) \tag{7.1}$$

where v = ei is an expression from a rule conclusion that sets v.

It is generally not possible to establish the truth of (7.1) directly. However, the user can build a formula that expresses the truth of (7.1) in terms of the input variables of the system. To build this formula, the user needs the following *hypothesis function* on atomic logical formulas in E:

Let X be a formula of the form,

VARIABLE = VALUE

If there is a rule in E containing X in its conclusion,

H(X) = OR(Hi(X))

where Hi is the hypothesis (if part) of a rule in E which contains X in its then part.

Otherwise H(X) = X.

Using the function H, one can define a logical formula that expresses (7.1.) in terms of input variables:

```
Let F0 be a variable over logical formulas.

F0 = (8.1);

while

 ( F0 contains an atomic formula for which H(X) != X)

F = the result of substituting H(X) for X in f;

return F0;
```

The resulting logical formula, which will be called COMPLETENESS, expresses completeness in terms of input variables to the expert system E. E is complete if the truth value of this formula is TRUE. To prove that COMPLETENESS is TRUE:

1. Write COMPLETENESS in conjunctive normal form.

2. Eliminate OR's containing logical opposites or all possible values of a variable.

If the resulting logical expression is TRUE, the system is complete. If the resulting logical expression is something else (call it COMPLETE0 for discussion purposes), then COMPLETE0 expresses the conditions under which the system produces a conclusion. Although not logically true, COMPLETE0 may be true because of mathematical theorems or domain knowledge.

Alternatively, NOT COMPLETE0 may be satisfiable. In this case, the expert system E is not complete.

Figure 7.1 below illustrates the above explanation of completeness.

Completeness of Investment Subsystem

To show the completeness of the investment subsystem (call it INV) of KB1, the first step is to construct the formula (7.1) for INV:

```
          investment = stocks OR investment = "bank account"                    (7.1.a)
```

Expressing this in terms of input conditions gives

```
          ( "Risk tolerance" = high                                             (7.1.b)
            AND "Discretionary income exists" = yes )
          OR
          ( "Risk tolerance" = low
            OR "Discretionary income exists" = no)
```

Writing this in conjunctive normal form gives

```
          ( "Risk tolerance" = high                                             (7.1.c)
            OR   "Risk tolerance" = low
            OR   "Discretionary income exists" = no )
          AND
          ( "Discretionary income exists" = yes
            OR   "Risk tolerance" = low
            OR   "Discretionary income exists" = no )
```

The first term is TRUE because high and low are the *only* possible values for risk tolerance. Likewise the second term is TRUE because yes and no are the only possible values for discretionary income exists. Therefore, the formula expressing completeness of INV is TRUE, and INV is complete.

Figure 7.1: Completeness of Investment Subsystem

Consistency

To verify consistency, the user must demonstrate that for all inputs, the expert system produces *a consistent set* of conclusions, i.e., that for each set of possible inputs, all the conclusions of the expert system can be true at the same time. (As noted in an earlier chapter, determining which sets of possible conclusions are consistent generally requires expert knowledge.)

To establish consistency, the user must do the following:

1. Construct a logical formula that represents conditions under which consistency fails; this logical formula will be called the *consistency formula.*

2. Show that the truth value of that formula is FALSE.

For a system E that is required to take one of some set of actions, a pseudo-variable is created whose values are the enumerated set of acceptable actions. Then the consistency formula for E is the consistency formula for a system that, perhaps among other things, outputs the value of this variable.

If there are no sets of inconsistent possible outputs, the system is consistent. Some expert systems are designed to recommend a set of components of a solution, and no one component contradicts any other. An investment advisor who recommended to each investor a set of desirable investments would be an example of this. For such systems, consistency is not an issue.

Let I1,...,In be the sets of mutually inconsistent possible conclusions of E. Each I consists of some set of conclusions, e.g.,

Ii = {Ci1, ... , Ci(mi)} (7.2)

where the Cs are possible conclusions of E.

The consistency formula for E is:

F(I1) or F(I2) ... or F(In) = FALSE (7.3)

where

F(Ii) = Ci1 and ... and Ci(mi) (7.4)

It is generally not possible to establish the truth of (7.4) directly. A formula can be built, however, that expresses the truth of (7.4) in terms of the input variables of the system. Just as for completeness, this formula is constructed by substituting the OR of rule hypotheses that infer a conclusion for that conclusion. Substituting the hypothesis function (7.2) into (7.4) using the iterative algorithm (7.4) constructs the consistency formula.

The resulting logical formula, which will be called CONSISTENCY, expresses consistency in terms of input variables to the expert system E. E is consistent if the truth value of this formula is TRUE. To prove that CONSISTENCY is TRUE:

1. Write CONSISTENCY in disjunctive normal form.

2. Eliminate ANDs containing logical opposites or other contradictory sets of conjuncts.

If the left hand side of the resulting logical expression is FALSE, the system is consistent. If the resulting logical expression is something else (call it CONSISTENT0 for discussion purposes); then CONSISTENT0 expresses the conditions under which the system produces possibly contradictory conclusions. Although not logically false, CONSISTENT0 may be false because of mathematical theorems or domain knowledge.

Alternatively, CONSISTENT0 may be satisfiable. In this case, the expert system E is not consistent, and is inconsistent for the inputs which satisfy CONSISTENT().

Consistency of Investment Subsystem:

To show the consistency of the investment subsystem (call it INV) of KB1, the first step is to construct the formula (7.2.3) for INV. The only set of inconsistent conclusions is:
{investment = stocks, investment = "bank account"} (7.2.a)
Therefore, (7.2.3) for INV is:

investment = stocks AND investment = "bank account" (7.2.b)

To show INV is consistent, one must show that (7.2.b) is FALSE.

Expressing this in terms of input conditions gives:

```
( "Risk tolerance" = high                                   (7.2.c)
  AND "Discretionary income exists" = yes )
AND
( "Risk tolerance" = low
  OR "Discretionary income exists" = no)
= FALSE
```

Writing this in disjunctive normal form gives:

```
( "Risk tolerance" = high                                   (7.1.d)
  AND "Discretionary income exists" = yes )
  AND  "Risk tolerance" = low )
OR
( "Risk tolerance" = high
  AND "Discretionary income exists" = yes )
  AND "Discretionary income exists" = no   )
= FALSE
```

The first term is FALSE because high and low are contradictory values for risk tolerance. Likewise the second term is FALSE because yes and no are contradictory values for discretionary income exists. Therefore, the left hand side of (7.1.d) is an OR of FALSE's, and is FALSE. This establishes the truth of the consistency formula for INV, and therefore INV is consistent.

Figure 7.2: Consistency of I Subsystem

Specification Satisfaction

While the vast range of possible specifications (as well as the Goedel Incompleteness Theorem makes it impossible to give a general method for proving specifications, there are some particular kinds of specifications where certain methods are useful.

Many valid specifications are not directly provable because they are not expressed in the variables and propositions used for the knowledge base. Before a specification can be proved it must be translated into the variables and relations used in the knowledge base. Translating specifications into the language of a knowledge base requires expert knowledge. Furthermore, this translation process may expose conditions under which the specifications are violated.

Step 1: Find all the possible conclusions that are constrained by the specification.

Step 2: Show that each of these conclusions are only made when permitted by the specification; i.e. for the specification S, and each conclusion C identified in Step 1,

If C then S(C) (7.5)

where S(C) is the result of substituting the variable values contained in C into S.

Let EC be the conditions under which the expert system E concludes C. EC is computed by procedure (7.5) above. Suppose:

EC implies S(C) (7.6)

Then whenever C occurs, i.e., when EC is true, S(C) is also true. On the other hand, if expert knowledge causes one to question (8.6), there is reason to think that the expert system can conclude C when S(C) is false.

Figure 7.3 shows a reasonable specification for Knowledge Base1.

A reasonable specification for KB1 is that it never recommend an unaffordable investment.
Step 1: The conclusion, investment = stock, is an investment that might not be affordable.
Step 2: Formulate how each conclusion is affected by the constraint, e.g.,
Expert system concludes "investment = stock" implies stock is affordable.
The successive substitutions of H(X) for X in this statement, using algorithm (7.4), produce a succession of ever more detailed statements about when the specification is true. For INV, these statements are:

```
If "Risk tolerance" = high
AND "Discretionary income exists" = yes
the stocks are affordable.
If ("Do you buy lottery tickets" = yes                    (7.3.a)
  OR "Do you currently own stocks" = yes)
AND
        ("Do you own a boat" = yes                        (7.3.b)
  OR "Do you own a luxury car" = yes)
THEN the stocks are affordable.
```

The truth of these statements depends on expert knowledge. If experts doubt *any* of them, it is probably because the conditions found in KB1 under which it concludes investment = stocks, are not sufficient to guarantee the specifications. In fact, (7.3.a) seems plausible while (7.3.b) seems weak. This indicates that the conditions for concluding the intermediate hypotheses,

If "Risk tolerance" = high
AND "Discretionary income exists" = yes

are not completely expressed in KB1.

Figure 7.3: Example Specification for KB1

Specification Based on Domain Subsets

Many specifications are of the form:

If the input is in some set S, (7.7)

then the output satisfies a logical formula P.

where S is defined by a logical formula C(I) over the input variables, and B(C1,...,Cn)) is a formula built over the conclusions, Ci, of the expert system; i.e. (7.7) becomes:

If C(I) then B(C1,...,Cn) (7.8)

To prove specifications like (7.8), *symbolic evaluation* of the knowledge base is a useful technique; the user can try to prove (7.8) by symbolic evaluation using either forward or backward chaining. With these proof methods, the user simulates the inference engine operating on inputs using the knowledge base. However, instead of actual input values, the only thing known about the inputs is that they satisfy C(I). This may be enough, however, to establish that the if a position of some of the rules are satisfied, then the conclusions will be derived within the said part of the rule. If so, these conclusions have been proved true on the basis of the assumptions, C(I). Further reasoning may lead eventually to showing that B is true.

Here is a forward chaining algorithm to prove B given C(I):

```
1. Assume C(I) = TRUE.  :                                            (7.9)
2. If the truth value of B can be established using known information,
       do so and goto X.
3. If the if part of a previously unsatisfied rule can be satisfied,
       then set the then part of the rule to TRUE and goto 2
       else quit, failing in the attempt to prove B.
4. If B is true,
       then the specification has been proved
       else if B is false
       then the specification is not satisfied.
```

Here is the backward chaining algorithm:

```
1. CURRENT = if C(I) then B(C1,...,Cn).                              (7.10)
2. If the truth value of CURRENT can be established, do so and quit with the following result:
       If CURRENT is true the specification has been proved,
       but if CURRENT is false, the specification is not true.
3. If there is an atomic formula A for which H(A) != A (see 7.2)
       substitute H(A) for A in CURRENT;
       Goto 2.
```

4. Quit with failure to establish the specification.

Figure 7.4 shows the symbolic evaluation of the KB1 example from chapter 5.

To illustrate symbolic evaluation, the following specification will be verified on the original KB1 of chapter 5:

If current savings < $3000, recommend that the investment is savings account. (7.4.a)

To prove the requirement, one assumes the condition:

"Savings balance" < $3000 (7.4.b)

and tries to prove that the expert system concludes that:

investment = "bank account". (7.4.c)

The strategy for carrying out this proof is to use a modification of the expert system's inference engine. It must be assumed that the inference engine makes inferences according to the rules of propositional logic. It is left to the scheduling strategy programmed into the inference engine to determine which of all the possible inferences that could be made are in fact made. For illustrative purposes, a backward chaining strategy is assumed.

Using a backward chaining strategy to prove that the expert system concludes 7.4.c, the user starts with that conclusion and shows that it is satisfied. The only way to do this in Knowledge Base 1B is to satisfy the "if" part of Rule 2. These conditions are true whenever

"Discretionary income exists" = no. (7.4.d)

Rule 6 makes this conclusion whenever:

"Savings balance" <= $3000. (7.4.e)

so (7.4.a) follows.

Notice that this proof mimics the inference engine of the expert system. In fact, every step of the proof could be carried out by the inference engine except for the last step of concluding (7.4.a). However, a modified inference engine could carry out that step if, whenever a truth value for an inequality was needed, a knowledge base about inequalities was consulted. In fact, such a knowledge base appears in the appendix of this chapter, and contains a rule that says

If X<=C then X<C.

Using this rule, a modified inference engine that consults an knowledge base about when atomic formulas are true is able to *automatically prove the desired condition* (7.4.a) *about the knowledge base*.

Figure 7.4: Symbolic Evaluation

The main differences between actual and symbolic inference engines are:

- Actual inference engines collect actual values for variables and use them in evaluating the conditions of rules to see if those rules fire.
- Symbolic inference engines have available logical conditions about the value of variables, e.g., the hypotheses C(I). Symbolic inference engines use this known information to infer whether atomic formulas in rule hypotheses are true or false. In an appendix to this chapter appear rules for symbolically determining the value of some arithmetic atomic formulas.

To construct a symbolic inference engine from an actual inference engine, the function that determines the truth of atomic formulas is replaced, but leaves the rest of the inference engine code intact. The *actual inference engine* determines the truth of atomic formulas by:

- Looking up the actual values of variables.
- Substituting them into the atomic formula.
- Determining the truth value of the result.

In a symbolic inference engine, the user can also evaluate atomic formulas when only *conditions about* variable values are known, but the actual values are unknown. The symbolic inference engine evaluates atomic formulas by:

- Assuming the known conditions about the variables in the knowledge base.
- Using this information to establish the truth of the atomic formula.

To build a symbolic inference engine requires the user to replace the function for the actual evaluation of atomic formulas with a function for symbolic evaluation, and to leave the rest of the inference alone.

Figure 7.5 list the steps involved for the actual inference engine.

Actual inference engine: For KB1, suppose the user said that his or her savings balance was $2000. Then the truth value of the atomic formula:

savings balance < $3000 (7.5.a)

can be determined by substituting in $2000 for "savings balance" to produce:

$2000 < $3000 (7.5.b)

This inequality is seen to be TRUE.

Symbolic inference engine: Suppose that:

"savings balance" <= $2000 (7.5.c)

is known to a symbolic inference engine. The symbolic inference engine tries to prove:

"savings balance" <= $2000 (7.5.d)
IMPLIES "savings balance" <= $3000

This formula is seen to be true. In fact, using the following row of the table in the appendix for evaluating atomic formulas,

ATOMIC FORMULA	TRUTH CONDITION	FALSE CONDITION
[a,b]=<c	b<=c	a>c

one may conclude (7.5.d).

Figure 7.5: Symbolic Inference Engine

Effect of the Inference Engine

The consistency and completeness techniques and the forward and backward symbolic inferences engines presented so far in this chapter are based on the assumption that fairly standard inference engines are used in processing the expert system knowledge base. These standard algorithms are capable of making all the inferences of propositional logic. Inference engines can depart from these algorithms by either error or design. For example, an inference may stop inference after the first knowledge base conclusion.

The most probable effect of departures in the inference algorithm from the standard is that the inference algorithm makes *fewer* inferences than the standard algorithm. An inference engine error of commission, a false inference, is more likely to be found during testing of the inference engine, while an omitted inference is harder to detect.

Consistency: Omitted inferences do not affect the above methods for finding consistency. The omissions merely mean that there are fewer conclusions to be inconsistent than expected.

Completeness: Omitted inferences *do* affect completeness. The omitted inferences may cause the inference engine not to make an expected conclusion. Where the inference engine is known to omit some propositional logic inferences, it is suggested that completeness be verified using a symbolic version of the same inference engine used on the knowledge base, incomplete though its inferences may be.

Satisfaction Of Specification: Specifications can be verified using symbolic versions of the *same* inference engine used to run the knowledge base. This provides the best insurance that the inference engine will actually make inferences that correspond to those made during the proof of the specification according to the rules of logic.

Inference Engines for Very High Reliability Applications

For applications where very high reliability is required, it is essential that the inference engine be known to make correct inferences. CLIPS (C Language Integrated Production Systems, first released by NASA in 1988) is the only expert system shell claimed to have a certified correct inference engine.

In order to know that an inference engine makes correct inferences, it is necessary that the inference engine be proven correct. One possible standard of correctness is that the inference engine makes all inferences possible with propositional logic given the information gathered from the user and other sources. This, or an alternative standard, should be proved by carrying out a formal correctness proof on the source code of the inference engine, practical only for small, simple ones.

On the basis of this or other proven description of inference engine behavior, algorithms can be constructed like those shown in this chapter for consistency, completeness, and symbolic evaluation for specification satisfaction.

In order to carry out in practice a correctness proof or equivalent description of the inference engine it is necessary that both the source code of the inference engine be available and short and simple enough to allow a proof to be carried out, given the primitive state of program correctness proofs.